A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots

Jeffrey Delmerico and Davide Scaramuzza

***Abstract*— Flying robots require a combination of accuracy and low latency in their state estimation in order to achieve stable and robust flight. However, due to the power and payload constraints of aerial platforms, state estimation algorithms must provide these qualities under the computational constraints of embedded hardware. Cameras and inertial measurement units (IMUs) satisfy these power and payload constraints, so visual-inertial odometry (VIO) algorithms are popular choices for state estimation in these scenarios, in addition to their ability to operate without external localization from motion capture or global positioning systems. It is not clear from existing results in the literature, however, which VIO algorithms perform well under the accuracy, latency, and computational constraints of a flying robot with onboard state estimation. This paper evaluates an array of publicly-available VIO pipelines (MSCKF, OKVIS, ROVIO, VINS-Mono, SVO+MSF, and SVO+GTSAM) on different hardware configurations, including several single-board computer systems that are typically found on flying robots. The evaluation considers the pose estimation accuracy, per-frame processing time, and CPU and memory load while processing the EuRoC datasets, which contain six degree of freedom (6DoF) trajectories typical of flying robots. We present our complete results as a benchmark for the research community.**

I. Introduction

Visual-inertial odometry (VIO) is currently applied to state estimation problems in a variety of domains, including autonomous vehicles, virtual and augmented reality, and flying robots. The field has reached a level of maturity such that many commercial products now utilize proprietary VIO algorithms, and there are several open-source software packages available that offer off-the-shelf visual pipelines that can be deployed on an end-user's system of choice.

The current research literature offers some comparative results on the performance of the popular VIO algorithms, but these typically consider only a subset of the existing algorithms, and almost always analyze their performance when running on powerful desktop or laptop computers with abundant computational resources. However, the physical constraints of flying robots limit the onboard computing power that is available, and thus these results do not accurately represent the performance of these algorithms for flying robot state estimation.

The motivation of this paper is to address this deficiency by performing a comprehensive evaluation of publicly-available VIO algorithms on hardware configurations that are typical of flying robot systems. We restrict the scope of this study to monocular VIO pipelines, since that is the minimal setup necessary for reliable state estimation, and is a popular choice for flying robots due to its low weight and power consumption, with respect to other sensor configurations. In particular, we consider the following pipelines:

- MSCKF - an extended Kalman Filter (EKF) orginally proposed in [1], but with many subsequent variations.

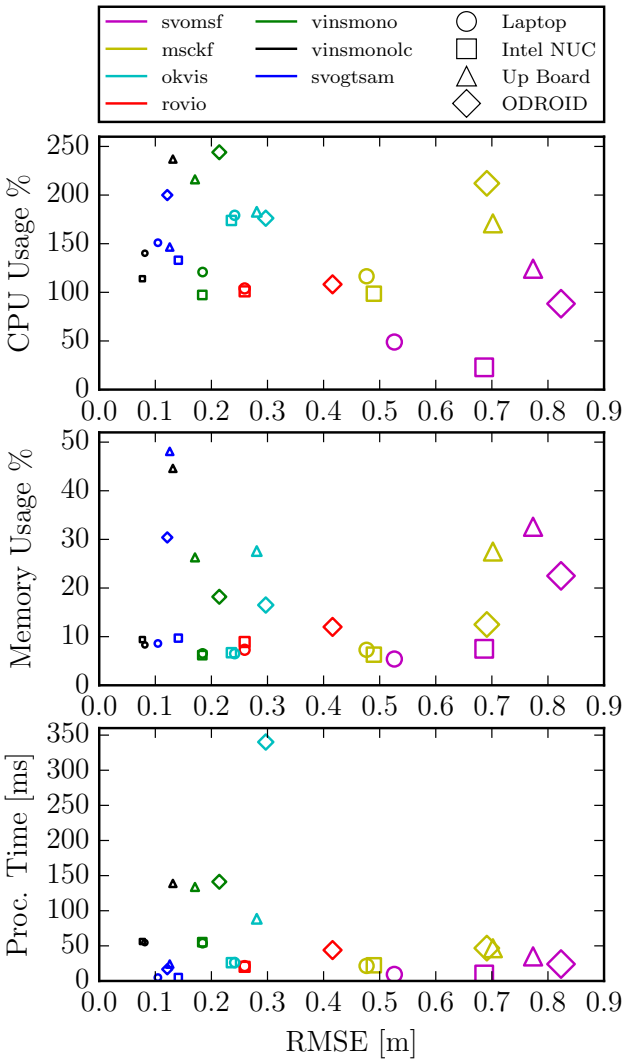

Fig. 1: Scatter plots showing algorithm efficiency, as measured by CPU utilization, memory usage, and per-frame processing time, versus RMS error. Each marker has a color representing its algorithm, shape representing its hardware platform, and a size that is proportional to the standard deviation of the error, summarized over all successful sequences in the EuRoC dataset.

This research was supported by the National Centre of Competence in Research (NCCR) Robotics, through the Swiss National Science Foundation, the SNSF-ERC Starting Grant, and the DARPA FLA program.

The authors are with the Robotics and Perception Group, Dep. of Informatics, University of Zurich , and Dep. of Neuroinformatics, University of Zurich and ETH Zurich, Switzerland—http://rpg.ifi.uzh.ch.

- OKVIS [2] - a keyframe- and optimization-based sliding window estimator using landmark reprojection errors.
- ROVIO [3] - an extended Kalman Filter with tracking of both 3D landmarks and image patch features.
- VINS-Mono [4] - a nonlinear-optimization-based sliding window estimator using pre-integrated IMU factors.
- SVO [5]+MSF [6] - a loosely-coupled configuration of a visual odometry pose estimator and an extended Kalman Filter for fusing the visual pose estimate with the inertial sensor data, as proposed in [7].
- SVO+GTSAM [8] - a lightweight visual odometry frontend with a full-smoothing backend provided by iSAM2 [9]

We do not consider non-inertial visual simultaneous localization and mapping (SLAM) systems, for example ORB-SLAM [10] and LSD-SLAM [11]. While these methods could potentially also be used for flying robot state estimation, we focus this benchmark on visual-inertial methods. In principle, one could pair one of these visual front ends with a Kalman Filter such as MSF [6] or with a pose graph optimization backend like iSAM2 [9] in order to incorporate inertial measurements, but such integrated systems are not publicly available.

Our experiments were conducted on the EuRoC Micro Aerial Vehicle datasets [12]. These sequences contain synchronized stereo camera and IMU data that was captured from a flying robot executing 6DoF motions in several indoor environments, with accurate ground truth provided by laser or motion capture tracking, depending on the sequence. These datasets have been used in many of the existing partial comparative results for VIO performance, and are currently the most extensive public set of image and IMU sequences for evaluating flying robot motion estimation algorithms.

One dimension that is not explored in this paper is a full design-space exploration for parameter optimization, which can have a dramatic effect on algorithm performance. While this could potentially be accomplished within a framework like SLAMBench [13] for dense RGB-D SLAM, currently no such framework for VIO algorithms exists. We have instead engaged the authors of each algorithm in order to utilize an optimal manual tuning of parameters, such that the performance is indicative of what could be obtained in a field deployment of a flying robot, where an offline search for optimal parameters for the environment is not possible.

Our goal is to provide a thorough benchmark of VIO pipelines on flying-robot-specific trajectories and hardware, in order to provide a reference for researchers on visual inertial odometry methods, as well as readers who require an off-the-shelf state estimation solution that is appropriate for their flying platform. Figure 1 summarizes the results of our evaluation in terms of the trade-off between accuracy and efficiency for all combinations of VIO algorithm and hardware platform.

A. Related Work

Within the current research literature, there is no benchmark study that satisfies our proposed goals. While comprehensive visual state estimation comparisons exist [14], they focus on only non-inertial methods and purely visual SLAM systems. Similarly, several benchmark datasets have been used for comparative studies of visual odometry algorithms, but these are either vision-only (e.g. TUM RGB-D [15], TUM monoVO [16], ICL-NUIM [17]), or contain non-6DoF trajectories (e.g. KITTI [18], Málaga [19]).

Since we focus on visual-inertial methods on flying robots, we can instead consider the existing results that are relevant to this problem. Among the methods evaluated in this paper, the recently proposed VINS-Mono pipeline [4] compares to OKVIS, but only on a few of the EuRoC datasets. In [2], OKVIS was compared to a non-public implementation of MSCKF [1] on non-public datasets. ROVIO [3] was evaluated with flying experiments, but was not compared to any other VIO methods. The visual odometry system that serves as the frontend for two of the methods considered in this paper, SVO [5], has been evaluated on the EuRoC datasets [12], but only compares to other non-inertial methods. The experiments in [8] compare the SVO + GTSAM system to OKVIS and a non-public implementation of MSCKF, but only on non-public datasets without flying-robot-like motions.

A number of other approaches to visual-inertial odometry have been proposed [20], [21], [22], [23], [24], [25], [26], but these do not offer publicly-available implementations. All of these are evaluated on some of the EuRoC datasets, but none contain a comprehensive comparison to other algorithms.

Many non-inertial visual odometry methods have been tested on the EuRoC datasets [12], including ORB-SLAM2 and LSD-SLAM [10], DSO [27], and a combined feature and direct approach [28]. However, these results are also incomplete, utilizing only a subset of the EuRoC datasets.

Most important, however, is that no existing work considers the additional dimension of computational constraints in their evaluation, instead only testing on full-featured computers or a single embedded system. This paper seeks to investigate the performance of VIO algorithms in real-world conditions by exploring this dimension and testing on a variety of hardware types that represent the computational resources available on a typical flying robot.

B. Contributions

This paper makes the following contributions:

- a comprehensive evaluation of publicly-available monocular visual-inertial odometry algorithms;
- comparative results for the performance of these algorithms on multiple embedded hardware platforms when processing 6DoF trajectories.

II. Visual-Inertial Odometry Algorithms

We next briefly summarize the primary features of the VIO algorithms represented in this benchmark. While these different approaches are not an exhaustive enumeration of the algorithms that have been proposed in the literature, the set of publicly-available implementations does cover the spectrum of approaches reasonably well, consisting of both loosely

and tightly coupled approaches, filtering and optimization-based algorithms, as well as several different variations on representing features and error terms.

A. MSCKF

The Multi-state constraint Kalman filter forms the basis of many modern, proprietary VIO systems, but until recently no sufficient, publicly available implementation existed. The original MSCKF algorithm in [1] proposed a measurement model that expressed the geometric constraints between all of the camera poses that observed a particular image feature, without the need to maintain an estimate of the 3D feature position in the state. The extended Kalman filter backend in [29] implements this formulation of the MSCKF for event-based camera inputs, but has been adapted to feature tracks from standard cameras. At the time of publication of this paper, this MSCKF implementation will be publicly available.[1].

B. OKVIS

Open Keyframe-based Visual-Inertial SLAM (OKVIS) [2] utilizes non-linear optimization on a sliding window of keyframe poses. The cost function is formulated with a combination of weighted reprojection errors for visual landmarks and weighted inertial error terms. The frontend uses a multi-scale Harris corner detector [30] to find features, and then computes BRISK descriptors [31] on them in order to perform data association between frames. Keyframes older than the sliding window are marginalized out of the states being estimated. OKVIS uses Google's *ceres solver* [32] to perform non-linear optimization. It should be noted that OKVIS is not optimized for monocular VIO, and in [2] it shows superior performance using a stereo configuration. The software is available in a ROS-compatible package.[2]

C. ROVIO

Robust Visual Inertial Odometry (ROVIO) [3] is a visual-inertial state estimator based on an extended Kalman Filter (EKF), which proposed several novelties. In addition to FAST corner features [33], whose 3D positions are parameterized with robot-centric bearing vectors and distances, multi-level patches are extracted from the image stream around these features. The patch features are tracked, warped based on IMU-predicted motion, and the photometric errors are used in the update step as innovation terms. Unlike OKVIS, ROVIO was developed as a monocular VIO pipeline, which should be noted when considering the results presented in Sec. III. The pipeline is available as an open-source software package.[3]

D. VINS-Mono

VINS-Mono [4] is a non-linear optimization-based sliding window estimator, tracking robust corner features [34], similar to OKVIS. However, VINS-Mono introduces several new features to this class of estimation framework. The authors propose a loosely-coupled sensor fusion initialization procedure to bootstrap the estimator from arbitrary initial states. IMU measurements are pre-integrated before being used in the optimization, and a tightly-coupled procedure for relocalization is proposed. VINS-Mono additionally features modules to perform 4DoF pose graph optimization and loop closure. Although we do not explicitly consider full SLAM systems, due to the tight integration of the loop closure module in VINS-Mono, we evaluate the performance of the algorithm both with and without this module activated (referred to in the experiments as *vinsmonolc* and *vinsmono*, respectively). The software is available in both a ROS-compatible PC version and an iOS implementation for state estimation on mobile devices.[4]

E. SVO+MSF

Multi-Sensor Fusion (MSF) [6] is a general EKF framework for fusing data from different sensors in a state estimate. Semi-Direct Visual Odometry (SVO) [5] is a computationally lightweight visual odometry algorithm that aligns images by tracking FAST corner [33] features and minimizing the photometric error of patches around them. This sparse alignment is then jointly optimized with the scene structure by minimizing the reprojection error of the features in a nonlinear least-squares optimization. The pose estimated from the vision-only SVO is provided to MSF as the output of a generic pose sensor, where it is then fused with the IMU data, as proposed in [7]. Due to the loose coupling of this setup, the scale of the pose must be at least approximately correct, requiring some bootstrapping from either manual initialization, or another sensor for estimating distance (e.g. a laser range sensor). Both MSF[5] and SVO[6] are publicly available, and communicate through a ROS interface. This system is referred to in the experiments as *svomsf*.

F. SVO+GTSAM

The same visual odometry frontend as in the SVO+MSF system has also been paired with a full-smoothing backend performing online factor graph optimization using iSAM2 [9]. In [8], the authors present results using this integrated system and propose the use of pre-integrated IMU factors in the pose graph optimization. Both components of this approach, SVO and the GTSAM 4.0 optimization toolbox[7] [35], are publicly available, however the integration of these into a single system is not currently public. This system is referred to in the experiments as *svogtsam*.

III. Experiments

The goal of the experiments described in this section is to evaluate the VIO pipelines from Sec. II in conditions that emulate state estimation for a flying robot. To accomplish

[1] https://github.com/daniilidis-group/msckf_mono
[2] https://github.com/ethz-asl/okvis_ros
[3] https://github.com/ethz-asl/rovio
[4] https://github.com/HKUST-Aerial-Robotics/VINS-Mono
[5] https://github.com/ethz-asl/ethzasl_msf
[6] http://rpg.ifi.uzh.ch/svo2.html
[7] https://bitbucket.org/gtborg/gtsam/

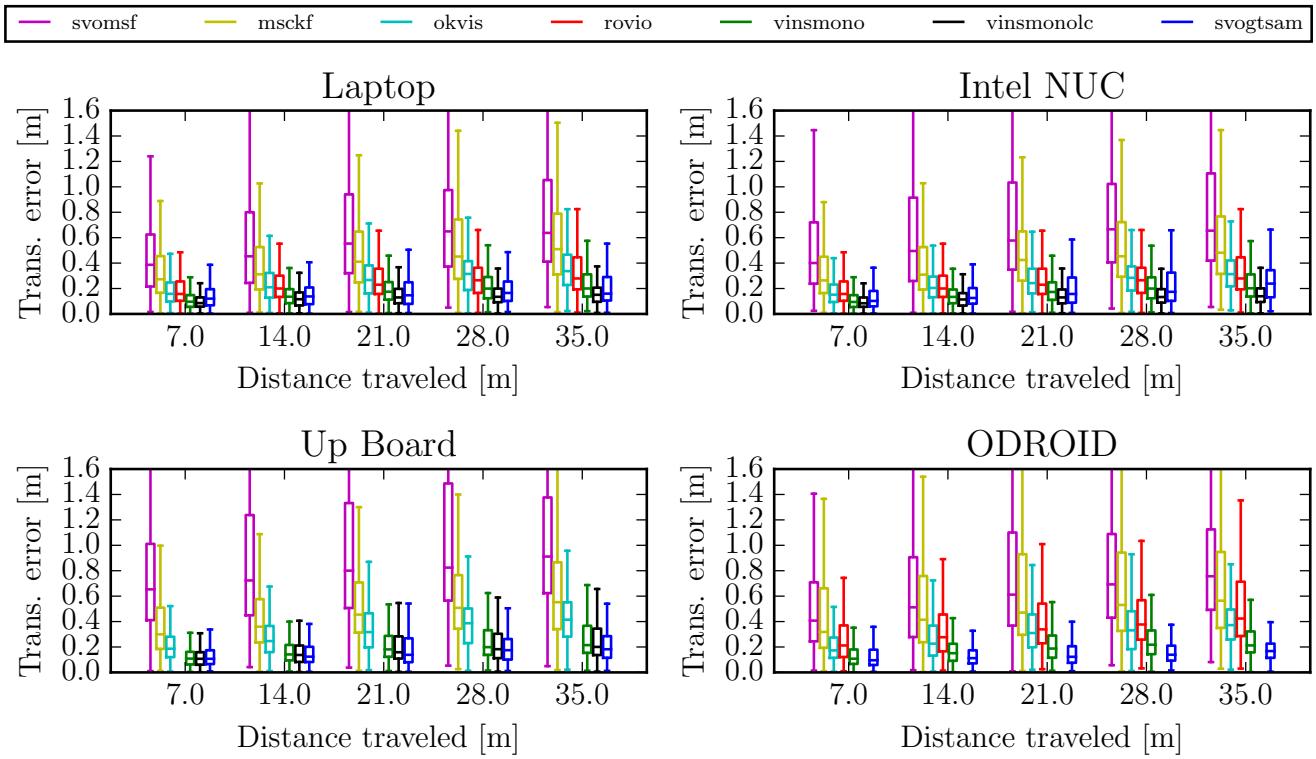

Fig. 2: Boxplot summarizing the translation error statistics for the VIO pipelines on each platform-algorithm combination over all dataset sequences. Errors were computed using the odometry metric from [18] over trajectory segments of lengths $\{7, 14, 21, 28, 35\}$ m.

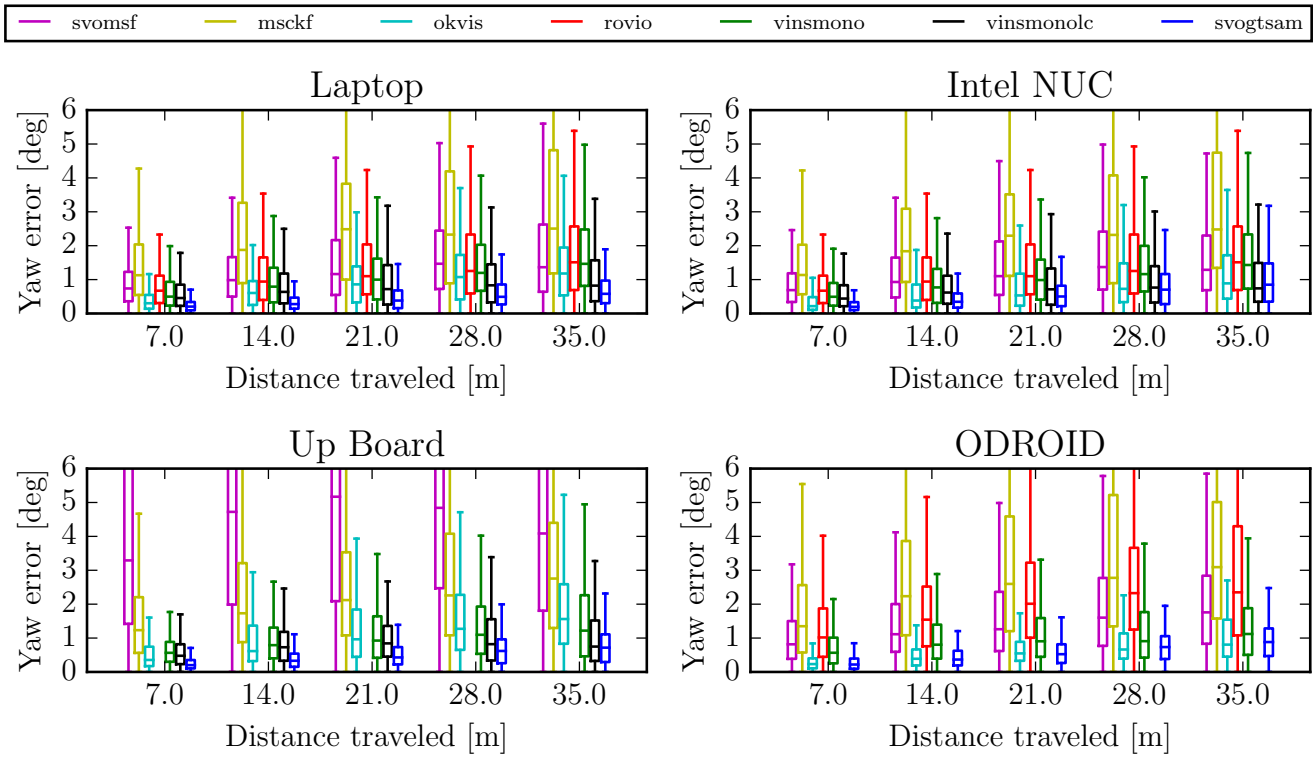

Fig. 3: Boxplot summarizing the statistics for angular error in yaw for the VIO pipelines on each platform-algorithm combination over all dataset sequences. Errors were computed using the odometry metric from [18] over trajectory segments of lengths $\{7, 14, 21, 28, 35\}$ m.

	Laptop							Intel NUC							Up Board							ODROID						
	svomsf	msckf	okvis	rovio	vinsmono	vinsmonolc	svogtsam	svomsf	msckf	okvis	rovio	vinsmono	vinsmonolc	svogtsam	svomsf	msckf	okvis	rovio	vinsmono	vinsmonolc	svogtsam	svomsf	msckf	okvis	rovio	vinsmono	vinsmonolc	svogtsam
MH 01	0.14	0.42	0.16	0.21	0.27	0.07	**0.05**	0.22	0.43	0.16	0.21	0.27	**0.07**	**0.08**	0.29	0.48	0.20	×	0.20	0.18	**0.12**	0.22	0.47	0.15	0.36	**0.13**	×	0.15
MH 02	0.20	0.45	0.22	0.25	0.12	0.05	**0.03**	0.20	0.43	0.20	0.25	0.12	**0.05**	**0.05**	0.31	0.40	0.22	×	0.18	0.19	**0.05**	0.24	0.63	0.20	0.23	0.08	×	**0.05**
MH 03	0.48	0.23	0.24	0.25	0.13	**0.08**	**0.12**	0.60	0.25	0.24	0.25	0.13	**0.05**	**0.12**	0.66	0.45	0.37	×	0.17	**0.09**	**0.10**	0.52	0.47	×	0.58	0.58	×	**0.12**
MH 04	1.38	0.37	0.34	0.49	0.23	**0.12**	**0.13**	1.82	0.61	**0.23**	0.49	**0.23**	**0.10**	0.24	2.02	0.67	0.44	×	**0.12**	0.15	0.24	2.28	0.64	0.42	0.81	**0.12**	×	×
MH 05	0.51	0.48	0.47	0.52	0.35	**0.09**	**0.16**	0.93	0.48	0.56	0.52	0.34	**0.11**	**0.16**	0.87	0.48	×	×	0.35	0.26	**0.13**	1.12	0.48	0.62	0.78	0.21	×	**0.12**
V1 01	0.40	0.34	0.09	0.10	**0.07**	**0.04**	**0.07**	0.39	0.29	0.09	0.10	**0.07**	**0.04**	0.12	0.36	0.25	0.13	×	**0.05**	**0.05**	0.08	0.43	0.21	0.09	0.15	0.11	×	**0.07**
V1 02	0.63	0.20	0.20	**0.10**	**0.10**	**0.06**	0.11	0.63	0.20	0.18	**0.10**	**0.10**	**0.05**	0.16	0.78	0.22	0.22	×	0.12	**0.08**	**0.10**	0.81	0.21	×	0.24	**0.11**	×	0.14
V1 03	×	0.67	0.24	0.14	**0.13**	**0.11**	×	×	0.67	0.20	0.14	**0.13**	**0.12**	×	×	0.63	0.30	×	**0.10**	**0.08**	×	×	1.52	×	0.20	**0.11**	×	×
V2 01	0.20	0.10	0.13	0.12	0.08	**0.06**	**0.07**	0.17	0.11	0.12	0.12	**0.08**	**0.06**	**0.08**	0.33	0.17	0.18	×	**0.08**	**0.05**	0.13	0.15	0.25	0.11	0.15	**0.08**	×	0.15
V2 02	0.37	0.16	0.16	0.14	**0.08**	**0.06**	×	0.37	0.16	0.17	0.14	**0.08**	**0.05**	×	0.59	0.18	0.30	×	**0.08**	**0.05**	×	0.46	0.19	0.26	0.17	**0.06**	×	×
V2 03	×	1.13	0.29	**0.14**	0.21	**0.09**	×	×	1.13	0.24	**0.14**	0.21	**0.09**	×	×	1.86	0.38	×	**0.17**	**0.09**	×	×	1.09	×	0.23	**0.16**	×	×

TABLE I: Absolute translation errors (RMSE) in meters for all trials. Errors have been computed after the estimated trajectories were aligned with the ground-truth trajectory using the method in [36]. The top performing algorithm on each platform and dataset is highlighted in **bold**. Trials where VINS-Mono with loop closure (*vinsmonolc*) achieves better accuracy than the best standard algorithm (without loop closure) are marked in **blue**.

this, we have selected a set of hardware platforms on which to run the algorithms, which represent the spectrum of computing resources that might be deployed on a flying system. We also utilize the EuRoC dataset [12], which is currently the most appropriate dataset for flying robot 6DoF trajectories.

A. Hardware Platforms

The hardware platforms that we consider include a desktop PC with a small form factor (Intel NUC), several single-board embedded computers (Up Board, ODROID), and a commodity laptop, which serves as a reference point for performance of the other systems, as well as providing a complete assessment of these algorithms on standard hardware, which only exists in the literature in piecemeal. We now briefly describe the technical specifications of the hardware platforms that we consider.

1) Laptop: This system is a Lenovo ThinkPad W540, a common mobile workstation. It has a quad-core Intel Core i7-4810MQ CPU with multi-threading, operating at 2.80GHz, and 32 GB of RAM. Due to the large form factor and high power requirements (nominally 47 W) of this system, it is not feasible for deployment on a flying robot, but is included here as a reference.

2) Intel NUC: The NUC is a small form factor desktop PC, which can be adapted to serve as a single-board computing system for mobile robots. It has a dual-core Intel Core i7-5557U CPU with multi-threading, operating at 3.10GHz, and 16 GB of RAM. The lower power requirements (28 W) and smaller size (10 x 10 cm) make the NUC a feasible option for embedded systems, while still providing comparable computing power to a commodity laptop.

3) UP Board: A 64-bit embedded single-board computer system, the UP Board contains a quad-core, single-threading Intel Atom x5-Z8350 CPU operating at 1.44GHz, with 4 GB of RAM. Its small size (8.5 x 5.6 cm), weight (78 g), and power consumption (12 W) make it appealing as a 64-bit embedded system for flying robots.

4) ODROID: The ODROID XU4 is an embedded PC containing a hybrid processing unit. The Samsung Exynos 5422 system on a chip consists of a quad-core ARM A7 at 1.5 GHz and a quad-core ARM A15 at 2.0 GHz in ARM's big.LITTLE configuration, allowing thread scheduling on one cluster of cores or the other, depending on CPU load. In addition, the ODROID has 2GB of RAM and has a similar form factor (8.3 x 5.8 cm) and power consumption (10 W) to the UP Board, but a smaller mass (59 g). Unlike the other systems, the ODROID uses a 32-bit architecture, but can make use of the NEON SIMD instruction set, while the 64-bit systems can use SSE instructions.

B. Datasets

The EuRoC MAV datasets [12] consist of eleven visual-inertial sequences recorded onboard a micro-aerial vehicle while it was manually piloted around three different indoor environments. Within each environment, the sequences increase qualitatively in difficulty with increasing sequence number. For example, Machine Hall 01 is "easy", while Machine Hall 05 is a more challenging sequence in the same environment, introducing things like faster motions, poor illumination, etc.

The sensor data was captured from a Visual-Inertial Sensor, which provides stereo WVGA monochrome images at 20 Hz, and temporally synchronized IMU data at 200 Hz. We use only the left camera image and IMU. In the Machine Hall sequences, ground truth positioning measurements were provided by a Leica MS50 laser tracker, while in the Vicon Room sequences they were provided by Vicon motion capture systems. The sequences, as well as the ground truth and sensor calibration data are publicly available.[8]

C. Evaluation

Each hardware platform was set up with Ubuntu 16.04 and ROS Kinetic, with the exception of the ODROID when running the OKVIS and VINS-Mono trials. Due to software

[8] http://projects.asl.ethz.ch/datasets/doku.php?id=kmavvisualinertialdatasets

incompatibilities, these two sets of tests needed to be run with Ubuntu 14.04 and ROS Indigo. Each VIO algorithm was configured so that it would process the sensor data as it was played back in real time, in order to simulate the live stream of sensor data that a flying robot would have available for estimating its state.

The authors of each algorithm provided recommended parameter settings, which were maintained across all trials. Although tuning parameters specifically for each sequence may improve performance, the goal of this comparison is to provide an assessment of the suitability of these algorithms and hardware platforms for use in general flying robot operations. The following exceptions to this general execution policy were either recommended by the algorithm authors or necessitated by computational constraints:

OKVIS On the Up Board and ODROID, in order to achieve real time performance, the maximum number of keypoints was reduced from 400 to 200, the keyframe window was reduced from 5 to 3, and the number of imu linked frames was reduced from 3 to 2.

ROVIO On the ODROID, the number of features was reduced from 25 to 10 in order to run successfully in real time. On the Up Board, no combination of parameters was found such that the filter converged successfully, even after attempts to reduce the computational load.

VINS-Mono The maximum number of tracked features was reduced from 150 to 100 in order to run in real time on the Up Board and ODROID. No parameter combination was successful in running VINS-Mono with loop closure activated on the ODROID.

SVO+MSF Due to the unobservability of visual scale in monocular visual odometry, it was necessary to bootstrap SVO with the correct scale by providing the ground truth poses during initialization. Once initialized, the poses then provided by SVO to MSF are of a scale consistent with the inertial measurements.

Additionally, compiler settings were set to the maximum level of optimization recommended by the algorithm authors, including application of SSE and NEON SIMD instructions wherever possible.

During each trial, the pose estimate was recorded after each state update from an input image. The time to process each image update, from receiving the image to updating the pose, was also recorded. The CPU and memory utilization were also sampled at a rate of 1 Hz during execution, measuring the total computational load in percentage of a single core, and percentage of total RAM allocated. If a VIO pipeline failed to initialize on a trial, it was restarted, and the first successful trial was taken.

For each trial, we performed *sim3* trajectory alignment to the ground truth according to the method from [36] and then computed the RMSE position error over the aligned trajectory. These results are shown in Table I, with the best performing algorithm highlighted in bold for each platform and sequence.

We also computed the odometric error using the metric in [18]. Differently from RMSE, using this metric, statistics about the accuracy of an algorithm are collected by aligning each estimated pose with its corresponding ground truth pose and then measuring the error in the estimate a fixed distance farther along the trajectory. In Figs. 2 and 3, we show statistics for the translation and yaw error accumulated over trajectory segments of lengths $\{7, 14, 21, 28, 35\}$ m[9] over all sequences for each platform-algorithm combination.

We similarly collect all of the CPU load, memory utilization, and time per frame measurements over all successful trials, and show them in Figs. 4, 5, and 6, respectively. These box and whisker plots show a box for the middle two quartiles, a line through the box for the median, and whiskers for the upper and lower quartiles.

IV. Discussion

An additional representation of the results in Sec. III is shown in Fig. 1. These scatter plots show the CPU usage, memory usage, and processing time per frame vs. RMSE, where each marker summarizes the performance of one algorithm on one hardware platform, over all of the EuRoC datasets. The markers are coded in color and shape by their algorithm and hardware platform, respectively, and illustrate the trade-offs between error and computational resources.

As the oldest algorithm considered in this evaluation, MSCKF still achieves competitive performance in some aspects of the benchmark. The algorithm was successful in completing all of the sequences on all of the hardware platforms, and the accuracy was consistent regardless of the platform. In addition to robustness, it generally provided modest resource usage and low per-frame processing time. However, most of the modern algorithms are able to achieve higher overall accuracy with a manageable increase in resource requirements.

OKVIS demonstrated accurate performance across all of the hardware platforms, including the embedded systems, despite low update rates there due to long per-frame processing times. This indicates that the underlying algorithm is robust, but the tolerance of low frame rate may be due in part to the low-speed trajectories in the EuRoC dataset.

While ROVIO exhibited tightly bounded and consistent resource usage, as well as accurate performance on all of the dataset sequences, it failed to run on the Up Board. On the other hardware platforms, the performance was accurate and consistent, suggesting good robustness to challenging trajectories, given a sufficiently powerful computer.

The performance of VINS-Mono was the most consistently accurate and robust across all of the hardware platforms. Enabling loop closure further improved the results, although this was not possible with the constrained computation available on the ODROID. This superior performance comes at the cost of a potentially prohibitive level of resource usage. In deploying this algorithm for state estimation on a flying robot, the user must consider the computational resources that will remain for navigation, control, and other perception applications.

[9]These evaluation distances were chosen based on the length of the shortest trajectory in the dataset, Vicon Room 2, at 36 m.

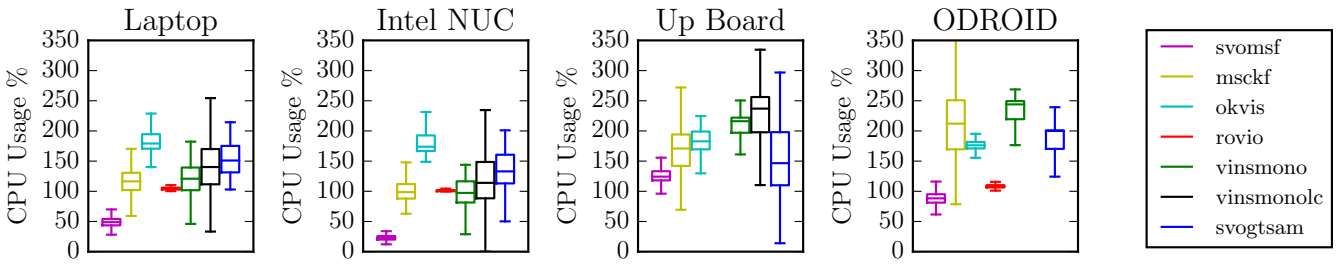

Fig. 4: CPU utilization statistics summarizing performance on all successful sequences for each platform-algorithm combination. Usage is represented as a percentage of a single CPU core on the given platform.

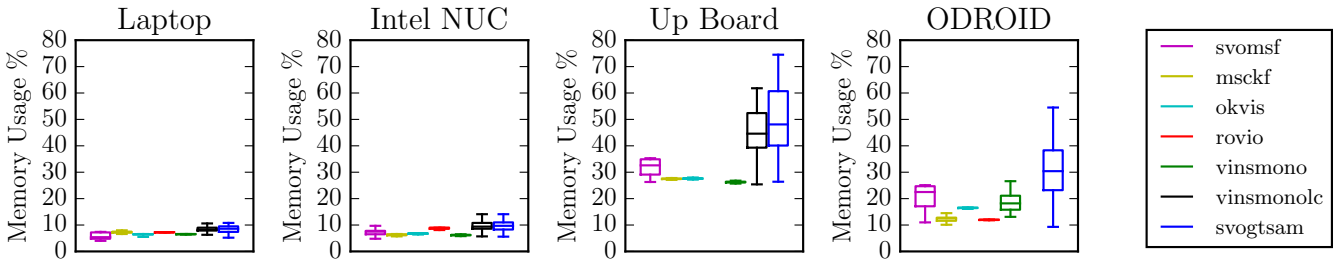

Fig. 5: Memory utilization statistics summarizing performance on all successful sequences for each platform-algorithm combination. Usage is represented as a percentage of the available RAM on the given platform.

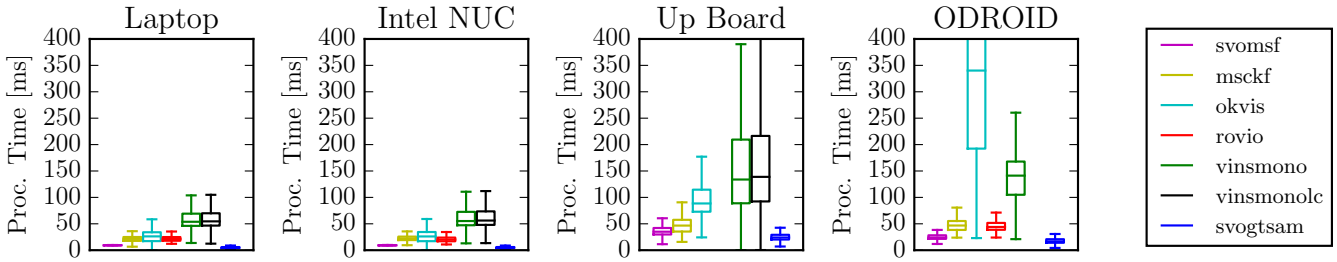

Fig. 6: Statistics for per-frame processing time, summarizing performance on all successful sequences for each platform-algorithm combination. Times were measured from the arrival of an input image until the state update for that image was completed, in milliseconds.

The only loosely-coupled pipeline considered here, SVO+MSF provides the highest level of computational efficiency, but with the corresponding lowest level of accuracy. This algorithm also required manual initialization in order to correctly estimate the scale of its pose, unlike the tightly-coupled approaches that estimate the scale directly in the state. However, for many flying robot applications, the level of accuracy that is possible with this configuration may be sufficient.

SVO+GTSAM produces the most accurate trajectories for many of the platform-dataset combinations, when considering the algorithms without explicit loop closing. It is able to accomplish this with relatively high CPU utilization, and high memory utilization, but with a consistently low frame processing time due to the decoupled frontend and backend. However, this approach is not as robust as other methods. One reason for this is that poorly triangulated visual features in the pose graph can cause numerical instabilities, causing the backend to fail. Consequently, despite some appealing properties for state estimation, this approach may not be appropriate for deployment on a flying robot.

The improvement in CPU utilization from the laptop to the NUC indicates that some of the algorithms, namely SVO+MSF, SVO+GTSAM, and VINS-Mono, are sensitive to CPU clock speed. If the algorithms run primarily in one compute-intensive thread, then a performance boost can be achieved by increasing clock speed. Further evidence for the importance of CPU clock speed is provided by the failure of ROVIO on the Up Board. Even with reduced parameter settings, the filter diverged quickly on all trials on this platform, suggesting that the update rate was too slow with the low clock rate of the Up Board's CPU, despite its otherwise sufficient computing resources.

These results indicate a few conclusions regarding the choice of state estimation algorithm for a flying robot system with an embedded single-board computer. Given a computationally constrained hardware platform like the Up Board or ODROID, SVO+MSF gives the most efficient performance, although it makes a significant sacrifice in overall accuracy, as well as robustness on challenging trajectories. If the

resource budget permits allocation of a significantly higher portion of computation to state estimation, then VINS-Mono (with loop closure if possible) provides the highest level of accuracy and robustness across all of the hardware platforms and sequences. A good compromise between these two extremes is ROVIO, which can provide better accuracy than SVO+MSF and much lower resource utilization than VINS-Mono. However, this comes with the caveat that it was not possible to run ROVIO on the Up Board, so the algorithm is more sensitive to per-frame processing time than the others.

V. Conclusion

In this work, we have conducted a survey of the state estimation performance of publicly-available visual-inertial odometry algorithms on hardware platforms with a range of computational resources. In evaluating these algorithms, our goal was to benchmark their performance on hardware and trajectories that are representative of state estimation for a flying robot with limited onboard computing power.

The results presented in Sec. III suggest that, as the reader may expect, there is no free lunch in visual state estimation. Accuracy and robustness can be improved with additional computation, but on systems with limited resources, finding the right balance between the competing requirements can be challenging. We hope that the results and conclusions presented in this paper may help members of the research community in finding appropriate compromises for their flying robot systems.

Acknowledgement

Thanks to Stefan Leutenegger, Michael Bloesch, and Tong Qin for their help in tuning OKVIS, ROVIO, and VINS-Mono, respectively. Special thanks to Kenneth Chaney, Alex Zhu, and Kostas Daniilidis for their assistance with the MSCKF code and experiments.

References

[1] A. I. Mourikis and S. I. Roumeliotis, "A multi-state constraint Kalman filter for vision-aided inertial navigation," in *ICRA*, 2007.

[2] S. Leutenegger, S. Lynen, M. Bosse, R. Siegwart, and P. Furgale, "Keyframe-based visual-inertial SLAM using nonlinear optimization," *Int. J. Robot. Research*, vol. 34, no. 3, pp. 314–334, 2015.

[3] M. Bloesch, S. Omari, M. Hutter, and R. Siegwart, "Robust visual inertial odometry using a direct EKF-based approach," in *IROS*, 2015.

[4] T. Qin, P. Li, and S. Shen, "VINS-Mono: A robust and versatile monocular visual-inertial state estimator," *arXiv preprint arXiv:1708.03852*, 2017.

[5] C. Forster, Z. Zhang, M. Gassner, M. Werlberger, and D. Scaramuzza, "SVO: Semidirect visual odometry for monocular and multicamera systems," *IEEE Trans. Robot.*, vol. 33, no. 2, pp. 249–265, 2017.

[6] S. Lynen, M. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A robust and modular multi-sensor fusion approach applied to MAV navigation," in *IROS*, 2013.

[7] M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3D mapping with a quadrotor MAV," *J. Field Robot.*, vol. 33, no. 4, pp. 431–450, 2016.

[8] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-manifold preintegration for real-time visual-inertial odometry," *IEEE Trans. Robot.*, vol. 33, no. 1, pp. 1–21, 2017.

[9] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the Bayes tree," *Int. J. Robot. Research*, vol. 31, pp. 217–236, Feb. 2012.

[10] R. Mur-Artal and J. D. Tardós, "ORB-SLAM2: An open-source SLAM system for monocular, stereo, and RGB-D cameras," *IEEE Trans. Robot.*, vol. 33, no. 5, pp. 1255–1262, 2017.

[11] J. Engel, J. Schöps, and D. Cremers, "LSD-SLAM: Large-scale direct monocular SLAM," in *ECCV*, 2014.

[12] M. Burri, J. Nikolic, P. Gohl, T. Schneider, J. Rehder, S. Omari, M. W. Achtelik, and R. Siegwart, "The EuRoC micro aerial vehicle datasets," *Int. J. Robot. Research*, vol. 35, pp. 1157–1163, 2015.

[13] L. Nardi, B. Bodin, M. Z. Zia, J. Mawer, A. Nisbet, P. H. J. Kelly, A. J. Davison, M. Luján, M. F. P. O'Boyle, G. Riley, N. Topham, and S. Furber, "Introducing SLAMBench, a performance and accuracy benchmarking methodology for SLAM," in *ICRA*, 2015.

[14] A. Quattrini Li, A. Coskun, S. M. Doherty, S. Ghasemlou, A. S. Jagtap, M. Modasshir, S. Rahman, A. Singh, M. Xanthidis, J. M. O'Kane, and I. Rekleitis, "Experimental comparison of open source vision-based state estimation algorithms," in *ISER*, 2017.

[15] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of RGB-D SLAM systems," in *IROS*, 2012.

[16] J. Engel, V. Usenko, and D. Cremers, "A photometrically calibrated benchmark for monocular visual odometry," *arXiv preprint arXiv:1607.02555*, 2016.

[17] A. Handa, T. Whelan, J. McDonald, and A. Davison, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," in *ICRA*, 2014.

[18] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *CVPR*, 2012.

[19] J.-L. Blanco, F.-A. Moreno, and J. Gonzalez-Jimenez, "The Málaga Urban Dataset: High-rate stereo and lidars in a realistic urban scenario," *Int. J. Robot. Research*, vol. 33, no. 2, pp. 207–214, 2014.

[20] R. Mur-Artal and J. D. Tardós, "Visual-inertial monocular slam with map reuse," *IEEE Robot. Autom. Lett.*, vol. 2, no. 2, pp. 796–803, 2017.

[21] R. Clark, S. Wang, H. Wen, A. Markham, and N. Trigoni, "VINet: Visual-inertial odometry as a sequence-to-sequence learning problem," in *AAAI*, 2017.

[22] M. K. Paul, K. Wu, J. A. Hesch, E. D. Nerurkar, and S. I. Roumeliotis, "A comparative analysis of tightly-coupled monocular, binocular, and stereo VINS," in *ICRA*, 2017.

[23] Y. Song, S. Nuske, and S. Scherer, "A multi-sensor fusion MAV state estimation from long-range stereo, IMU, GPS and barometric sensors," *Sensors*, vol. 17, no. 1, 2017.

[24] A. Solin, S. Cortes, E. Rahtu, and J. Kannala, "PIVO: Probabilistic inertial-visual odometry for occlusion-robust navigation," *arXiv preprint arXiv:1708.00894*, 2017.

[25] S. Houben, J. Quenzel, N. Krombach, and S. Behnke, "Efficient multi-camera visual-inertial slam for micro aerial vehicles," in *IROS*, 2016.

[26] K. Eckenhoff, P. Geneva, and G. Huang, "Direct visual-inertial navigation with analytical preintegration," in *ICRA*, 2017.

[27] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PP, no. 99, pp. 1–1, 2017.

[28] N. Krombach, D. Droeschel, and S. Behnke, "Combining feature-based and direct methods for semi-dense real-time stereo visual odometry," in *Int. Conf. Intell. Auton. Sys.*, 2017, pp. 855–868.

[29] A. Zhu, N. Atanasov, and K. Daniilidis, "Event-based visual inertial odometry," in *CVPR*, 2017.

[30] C. Harris and M. Stephens, "A combined corner and edge detector," in *Proc. Alvey Vision Conf.*, 1988, pp. 147–151.

[31] S. Leutenegger, M. Chli, and R. Siegwart, "BRISK: Binary robust invariant scalable keypoints," in *ICCV*, 2011.

[32] A. Agarwal, K. Mierle, and Others, "Ceres solver," http://ceres-solver.org.

[33] E. Rosten, R. Porter, and T. Drummond, "Faster and better: A machine learning approach to corner detection," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 32, no. 1, pp. 105–119, Jan. 2010.

[34] J. Shi and C. Tomasi, "Good features to track," in *CVPR*, 1994.

[35] F. Dellaert, "Factor graphs and GTSAM: A hands-on introduction," Georgia Institute of Technology, Tech. Rep. GT-RIM-CP&R-2012-002, Sep. 2012.

[36] S. Umeyama, "Least-squares estimation of transformation parameters between two point patterns," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 13, no. 4, 1991.